Кортежи

static void Main(string[] args)

{

(int, string, char, string, ulong) cort = (789, "45r34", 'j', "dgdfmgk", 6748);

Console.WriteLine(cort);

Console.WriteLine($"{cort.Item1} {cort.Item3} {cort.Item4}");

{

(int a, string b, char c, string d, ulong e) = cort;

}

{

int a; string b; char c; string d; ulong e;

(a, b, c, d, e) = cort;

}

int MMM\_ = 644;

(int, string) cort1 = (5, "4h");

(int, string) cort2 = (5, "4h");

if (cort1 == cort2) Console.WriteLine("Кортежи одинаковы!");

}

static void Main(string[] args)

{

dynamic LocFun(int[] Arr, string str)

{

(int, int, int, string) cort = (Arr.Max(), Arr.Min(), Arr.Sum(), str.Substring(0, 1));

return cort;

}

int[] ArrToParm = new int[] { 423, 65, 654, 5432, 324 };

(int, int, int, string) RetCort = LocFun(ArrToParm, "Привет, пошли ко мне :)");

Console.WriteLine($"Максимальный элемент: {RetCort.Item1}");

Console.WriteLine($"Минимальный элемент: {RetCort.Item2}");

Console.WriteLine($"Сумма всех элементов: {RetCort.Item3}");

Console.WriteLine($"Первый символ строки: {RetCort.Item4}");

}

StringBuilder

StringBuilder sb = new StringBuilder("Привет, я xxx Илья");

Console.WriteLine(sb.ToString());

sb.Remove(10, 4);

sb.Insert(0, "XXX ");

sb.Append(" XXX");

Console.WriteLine(sb.ToString());

**Преобразования типов**  
Console.WriteLine("short");

short \_short = Convert.ToInt16(Console.ReadLine());

Console.WriteLine("ushort");

ushort \_ushort = Convert.ToUInt16(Console.ReadLine());

Console.WriteLine("long");

long \_long = Convert.ToInt64(Console.ReadLine());

Console.WriteLine("ulong");

ulong \_ulong = Convert.ToUInt64(Console.ReadLine());

Console.WriteLine("int");

int \_int = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("uint");

uint \_uint = Convert.ToUInt32(Console.ReadLine());

Console.WriteLine("float");

float \_float = Convert.ToSingle(Console.ReadLine());

Console.WriteLine("double");

double \_double = Convert.ToDouble(Console.ReadLine());

Console.WriteLine("decimal");

decimal \_decimal = Convert.ToDecimal(Console.ReadLine());

Console.WriteLine("char");

char \_char = Convert.ToChar(Console.ReadLine());

Console.WriteLine("byte");

byte \_byte = Convert.ToByte(Console.ReadLine());

Console.WriteLine("sbyte");

sbyte \_sbyte = Convert.ToSByte(Console.ReadLine());

Console.WriteLine("bool");

bool \_bool = Convert.ToBoolean(Console.ReadLine());

Console.WriteLine("string");

string \_string = Console.ReadLine();

Book book1 = new Book();

Console.WriteLine("object");

book1.cost = Console.ReadLine();

Console.WriteLine(book1.cost);

Console.WriteLine("short: {0} ushort: {1} \n long: {2} ulong: {3}\n int: {4} uint: {5}\n float: {6} double: {7}" +

"\ndecimal: {8} char: {9} \nbyte: {10} sbyte: {11} \nbool: {12} string: {13}", \_short, \_ushort, \_long, \_ulong,

\_int, \_uint, \_float, \_double, \_decimal, \_char, \_byte, \_sbyte, \_bool, \_string);

//Console.WriteLine("неявные преобразования");

**Нулл**  
bool? \_null1 = null;

Nullable<bool> \_null2 = null;

**Упаковка\распаковка**

int point = 6;

byte a1;

short a2;

uint a3;

a1 = (byte)point;

a2 = (short)point;

a3 = (uint)point;

//3

int box = 5;

object obj = box; // присваивание сопровождается упаковкой

int Unbox = (int)obj; // приведение вызовет распаковку

**Операции со строками**

text = text.Replace("хороший", "плохой");

// индекс последнего символа

int ind = text.Length - 1;

// вырезаем последний символ

text = text.Remove(ind);

// вставка  
text = text.Insert(8, subString);

// Разделение  
string[] words = text.Split(new char[] { ' ' });

// Поиск  
string s1 = "hello world";

char ch = 'o';

int indexOfChar = s1.IndexOf(ch); // равно 4

// сравнение  
int result = String.Compare(s1, s2);  
//соединение  
string s4 = String.Concat(s3, "!!!"); // результат: строка "hello world!!!"

**Перегрузка**

public static bool operator >(Variant23 first, Variant23 second )

{

return first.length > second.length;

}

**Переопределение**

public override int GetHashCode()

{

return pname.GetHashCode();

}

public override bool Equals(object obj)

{

if (obj.GetType() != this.GetType()) return false;

produser person = (produser)obj;

return (this.pname == person.pname);

}

**Исключения**

class PersonException : Exception

{

public PersonException(string message)

: base(message)

{ }

}

public string Type

{

get

{

return type;

}

set

{

if (value != "мюзикл")

throw new PersonException("Это музыкальный канал");

else

type = value;

}

}

catch (DivideByZeroException ex)

{

Console.WriteLine(ex.Message);

}

LINQ

Select: определяет проекцию выбранных значений

Where: определяет фильтр выборки

OrderBy: упорядочивает элементы по возрастанию

OrderByDescending: упорядочивает элементы по убыванию

ThenBy: задает дополнительные критерии для упорядочивания элементов возрастанию

ThenByDescending: задает дополнительные критерии для упорядочивания элементов по убыванию

Join: соединяет две коллекции по определенному признаку

GroupBy: группирует элементы по ключу

ToLookup: группирует элементы по ключу, при этом все элементы добавляются в словарь

GroupJoin: выполняет одновременно соединение коллекций и группировку элементов по ключу

Reverse: располагает элементы в обратном порядке

All: определяет, все ли элементы коллекции удовлятворяют определенному условию

Any: определяет, удовлетворяет хотя бы один элемент коллекции определенному условию

Contains: определяет, содержит ли коллекция определенный элемент

Distinct: удаляет дублирующиеся элементы из коллекции

Except: возвращает разность двух коллекцию, то есть те элементы, которые создаются только в одной коллекции

Union: объединяет две однородные коллекции

Intersect: возвращает пересечение двух коллекций, то есть те элементы, которые встречаются в обоих коллекциях

Count: подсчитывает количество элементов коллекции, которые удовлетворяют определенному условию

Sum: подсчитывает сумму числовых значений в коллекции

Average: подсчитывает cреднее значение числовых значений в коллекции

Min: находит минимальное значение

Max: находит максимальное значение

Take: выбирает определенное количество элементов

Skip: пропускает определенное количество элементов

TakeWhile: возвращает цепочку элементов последовательности, до тех пор, пока условие истинно

SkipWhile: пропускает элементы в последовательности, пока они удовлетворяют заданному условию, и затем возвращает оставшиеся элементы

Concat: объединяет две коллекции

Zip: объединяет две коллекции в соответствии с определенным условием

First: выбирает первый элемент коллекции

FirstOrDefault: выбирает первый элемент коллекции или возвращает значение по умолчанию

Single: выбирает единственный элемент коллекции, если коллекция содердит больше или меньше одного элемента, то генерируется исключение

SingleOrDefault: выбирает первый элемент коллекции или возвращает значение по умолчанию

ElementAt: выбирает элемент последовательности по определенному индексу

ElementAtOrDefault: выбирает элемент коллекции по определенному индексу или возвращает значение по умолчанию, если индекс вне допустимого диапазона

Last: выбирает последний элемент коллекции

LastOrDefault: выбирает последний элемент коллекции или возвращает значение по умолчанию

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Runtime.InteropServices;

using System.Text;

using System.Threading.Tasks;

namespace Lab10

{

public partial class DoubleStack

{

public void PushEl(double val)

{

NumbEl++;

if (CheckSize())

{

Console.WriteLine("Не удалось добавить элемент в стек. Стек переполнен!");

return;

}

stack.Push(val);

}

public void ShowEl()

{

//if (NumbEl == 0) { Console.WriteLine("Стек пуст!"); return; };

Console.WriteLine(stack.Peek());

}

public double GetEl()

{

double val = stack.Peek();

return val;

}

public void DelEl()

{

if (NumbEl == 0) { Console.WriteLine("Стек пуст!"); return; }

Console.WriteLine($"Элемент успешно удалён! Удалённый элемент: {stack.Peek()}");

stack.Pop();

NumbEl--;

}

public void ShowAll()

{

for (int i = 0; i < stack.Count; i++)

{

Console.Write(stack.ElementAt(i) + " ");

}

}

public bool CheckNegativeNumb()

{

for (int i = 0; i < stack.Count(); i++)

{

if (stack.ElementAt(i) < 0) return true;

}

return false;

}

public double GetSum { get { double val = stack.Sum(); return val; } }

}

internal class Program

{

static void Main(string[] args)

{

string[] Months = { "January", "February", "March", "April", "May", "June", "July", "August", "September", "October", "November", "December" };

int NumberOfSymbols = int.Parse(Console.ReadLine());

IEnumerable<string> CurrentMonths = from n in Months

where n.Length == NumberOfSymbols

select n;

foreach (string Month in CurrentMonths)

{

Console.WriteLine(Month);

}

Console.ReadKey();

Console.WriteLine();

CurrentMonths = from n in Months

where n == Months[0] || n == Months[1] || n == Months[11] || n == Months[5] || n == Months[6] || n == Months[7]

select n;

foreach (string Month in CurrentMonths)

{

Console.WriteLine(Month);

}

Console.ReadKey();

Console.WriteLine();

CurrentMonths = from n in Months

orderby n

select n;

foreach (string Month in CurrentMonths)

{

Console.WriteLine(Month);

}

Console.ReadKey();

Console.WriteLine();

CurrentMonths = from n in Months

where n.Contains('u') && n.Length == 4

select n;

foreach (string Month in CurrentMonths)

{

Console.WriteLine(Month);

}

Console.ReadKey();

Console.WriteLine();

List<DoubleStack> stacks = new List<DoubleStack>()

{

new DoubleStack(5, 35, 32),

new DoubleStack(12, 54,-534),

new DoubleStack(3, 55.34232, 23435, -546, 34),

new DoubleStack(9, 12, 4354, 0),

new DoubleStack(14),

new DoubleStack(243, -54),

new DoubleStack(1324, 345,3456, 46, 5654, 68.9824),

new DoubleStack(7, 43, -546),

new DoubleStack(1022, 32, 54.24),

new DoubleStack(9),

};

var OurStacks = stacks.Where(min => min.GetEl() == stacks.Min(minEl => minEl.GetEl()));

DoubleStack SelectStack = OurStacks.First();

SelectStack.ShowAll();

Console.WriteLine();

OurStacks = stacks.Where(max => max.GetEl() == stacks.Max(MaxEl => MaxEl.GetEl()));

SelectStack = OurStacks.First();

SelectStack.ShowAll();

Console.WriteLine();

Console.ReadKey();

Console.WriteLine();

OurStacks = stacks.Where(negative => negative.CheckNegativeNumb());

foreach (DoubleStack negative in OurStacks)

{

negative.ShowAll();

Console.WriteLine();

}

Console.WriteLine();

Console.ReadKey();

OurStacks = stacks.Where(st => st.size == 1 || st.size == 3);

DoubleStack[] ArrayStacks = new DoubleStack[OurStacks.Count()];

for (int i = 0; i < ArrayStacks.Length; i++)

{

ArrayStacks[i] = OurStacks.ElementAt(i);

ArrayStacks[i].ShowAll();

Console.WriteLine();

}

Console.WriteLine();

Console.ReadKey();

Console.WriteLine();

OurStacks = stacks.Where(zeroStacks => zeroStacks.GetEl() == 0).Take(1);

SelectStack = OurStacks.First();

SelectStack.ShowAll();

Console.WriteLine();

Console.ReadKey();

Console.WriteLine();

OurStacks = stacks.OrderBy(st => st.GetSum);

foreach (DoubleStack st in OurStacks)

{

st.ShowAll();

Console.WriteLine();

}

Console.WriteLine();

Console.ReadKey();

OurStacks = stacks.Where(st => st.CheckNegativeNumb()).OrderBy(st => st.GetEl()).Take(4).Select(st => st);

foreach (DoubleStack st in OurStacks)

{

st.ShowAll();

Console.WriteLine();

}

var MonthStack = from Month in Months

join stack in stacks on Month.Length equals stack.size

select new { Month, stack };

foreach (var item in MonthStack)

{

Console.WriteLine(item.Month + " - " + item.stack.size);

}

}

}

}

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Runtime.InteropServices;

using System.Text;

using System.Threading.Tasks;

namespace Lab10

{

public partial class DoubleStack

{

public void PushEl(double val)

{

NumbEl++;

if (CheckSize())

{

Console.WriteLine("Не удалось добавить элемент в стек. Стек переполнен!");

return;

}

stack.Push(val);

}

public void ShowEl()

{

//if (NumbEl == 0) { Console.WriteLine("Стек пуст!"); return; };

Console.WriteLine(stack.Peek());

}

public double GetEl()

{

double val = stack.Peek();

return val;

}

public void DelEl()

{

if (NumbEl == 0) { Console.WriteLine("Стек пуст!"); return; }

Console.WriteLine($"Элемент успешно удалён! Удалённый элемент: {stack.Peek()}");

stack.Pop();

NumbEl--;

}

public void ShowAll()

{

for (int i = 0; i < stack.Count; i++)

{

Console.Write(stack.ElementAt(i) + " ");

}

}

public bool CheckNegativeNumb()

{

for (int i = 0; i < stack.Count(); i++)

{

if (stack.ElementAt(i) < 0) return true;

}

return false;

}

public double GetSum { get { double val = stack.Sum(); return val; } }

}

internal class Program

{

static void Main(string[] args)

{

string[] Months = { "January", "February", "March", "April", "May", "June", "July", "August", "September", "October", "November", "December" };

int NumberOfSymbols = int.Parse(Console.ReadLine());

IEnumerable<string> CurrentMonths = from n in Months

where n.Length == NumberOfSymbols

select n;

foreach (string Month in CurrentMonths)

{

Console.WriteLine(Month);

}

Console.ReadKey();

Console.WriteLine();

CurrentMonths = from n in Months

where n == Months[0] || n == Months[1] || n == Months[11] || n == Months[5] || n == Months[6] || n == Months[7]

select n;

foreach (string Month in CurrentMonths)

{

Console.WriteLine(Month);

}

Console.ReadKey();

Console.WriteLine();

CurrentMonths = from n in Months

orderby n

select n;

foreach (string Month in CurrentMonths)

{

Console.WriteLine(Month);

}

Console.ReadKey();

Console.WriteLine();

CurrentMonths = from n in Months

where n.Contains('u') && n.Length == 4

select n;

foreach (string Month in CurrentMonths)

{

Console.WriteLine(Month);

}

Console.ReadKey();

Console.WriteLine();

List<DoubleStack> stacks = new List<DoubleStack>()

{

new DoubleStack(5, 35, 32),

new DoubleStack(12, 54,-534),

new DoubleStack(3, 55.34232, 23435, -546, 34),

new DoubleStack(9, 12, 4354, 0),

new DoubleStack(14),

new DoubleStack(243, -54),

new DoubleStack(1324, 345,3456, 46, 5654, 68.9824),

new DoubleStack(7, 43, -546),

new DoubleStack(1022, 32, 54.24),

new DoubleStack(9),

};

var OurStacks = stacks.Where(min => min.GetEl() == stacks.Min(minEl => minEl.GetEl()));

DoubleStack SelectStack = OurStacks.First();

SelectStack.ShowAll();

Console.WriteLine();

OurStacks = stacks.Where(max => max.GetEl() == stacks.Max(MaxEl => MaxEl.GetEl()));

SelectStack = OurStacks.First();

SelectStack.ShowAll();

Console.WriteLine();

Console.ReadKey();

Console.WriteLine();

OurStacks = stacks.Where(negative => negative.CheckNegativeNumb());

foreach (DoubleStack negative in OurStacks)

{

negative.ShowAll();

Console.WriteLine();

}

Console.WriteLine();

Console.ReadKey();

OurStacks = stacks.Where(st => st.size == 1 || st.size == 3);

DoubleStack[] ArrayStacks = new DoubleStack[OurStacks.Count()];

for (int i = 0; i < ArrayStacks.Length; i++)

{

ArrayStacks[i] = OurStacks.ElementAt(i);

ArrayStacks[i].ShowAll();

Console.WriteLine();

}

Console.WriteLine();

Console.ReadKey();

Console.WriteLine();

OurStacks = stacks.Where(zeroStacks => zeroStacks.GetEl() == 0).Take(1);

SelectStack = OurStacks.First();

SelectStack.ShowAll();

Console.WriteLine();

Console.ReadKey();

Console.WriteLine();

OurStacks = stacks.OrderBy(st => st.GetSum);

foreach (DoubleStack st in OurStacks)

{

st.ShowAll();

Console.WriteLine();

}

Console.WriteLine();

Console.ReadKey();

OurStacks = stacks.Where(st => st.CheckNegativeNumb()).OrderBy(st => st.GetEl()).Take(4).Select(st => st);

foreach (DoubleStack st in OurStacks)

{

st.ShowAll();

Console.WriteLine();

}

var MonthStack = from Month in Months

join stack in stacks on Month.Length equals stack.size

select new { Month, stack };

foreach (var item in MonthStack)

{

Console.WriteLine(item.Month + " - " + item.stack.size);

}

}

}

}

Рефлексия

using System;

using System.Collections;

using System.Collections.Generic;

using System.Dynamic;

using System.IO;

using System.Linq;

using System.Reflection;

using System.Text;

using System.Threading.Tasks;

namespace Lab11

{

static class Reflector

{

private static string fullInfo = "";

public static void WriteToFile()

{

StreamWriter sw = new StreamWriter("D:\\лабы ООП\\Lab11\\Info.txt");

sw.WriteLine(fullInfo);

sw.Close();

}

public static string ReadFromFile()

{

string text;

StreamReader sr = new StreamReader("D:\\лабы ООП\\Lab11\\Param.txt");

text = sr.ReadLine();

sr.Close();

return text;

}

public static void BuildInf()

{

string info = AssemblyName.GetAssemblyName("Lab11.exe").Name;

Console.WriteLine(info);

fullInfo += info + "\n\n";

}

public static void ConstrInf(string ClassName)

{

Type myType = Type.GetType("Lab11." + ClassName);

Console.WriteLine("Публичные конструкторы: ");

fullInfo += "Публичные конструкторы:\n";

foreach (ConstructorInfo ctor in myType.GetConstructors())

{

if (ctor.IsPublic)

{

Console.WriteLine(ctor.Name);

fullInfo += ctor.Name + "\n\n";

}

}

Console.WriteLine();

}

public static void MethodInf(string ClassName)

{

Type myType = Type.GetType("Lab11." + ClassName);

Console.WriteLine("Публичные методы: ");

fullInfo += "Публичные методы:\n";

foreach (MethodInfo method in myType.GetMethods())

{

if (method.IsPublic)

{

Console.WriteLine(method.Name);

fullInfo += method.Name + "\n";

}

}

Console.WriteLine();

fullInfo += "\n";

}

public static void FieldInf(string ClassName)

{

Type myType = Type.GetType("Lab11." + ClassName);

Console.WriteLine("Поля: ");

fullInfo += "Поля:\n";

foreach (FieldInfo field in myType.GetFields())

{

Console.WriteLine(field.Name);

fullInfo += field.Name + "\n\n";

}

Console.WriteLine();

}

public static void ProptInf(string ClassName)

{

Type myType = Type.GetType("Lab11." + ClassName);

Console.WriteLine("Свойства: ");

fullInfo += "Свойства:\n";

foreach (PropertyInfo property in myType.GetProperties())

{

Console.WriteLine(property.Name);

fullInfo += property.Name + "\n\n";

}

Console.WriteLine();

}

public static void InterInf(string ClassName)

{

Type myType = Type.GetType("Lab11." + ClassName);

Console.WriteLine("Реализованные интерфейсы:");

fullInfo += "Реализованные интерфейсы:\n";

foreach (Type inter in myType.GetInterfaces())

{

Console.WriteLine(inter.Name);

fullInfo += inter.Name + "\n\n";

}

Console.WriteLine();

}

public static void MethodForParamInf(string ClassName)

{

fullInfo += "Метод, найденный по параметру\n";

Type myType = Type.GetType("Lab11." + ClassName);

Console.Write("Введите тип параметра: ");

string param = Console.ReadLine();

foreach (MethodInfo method in myType.GetMethods())

{

ParameterInfo[] parameters = method.GetParameters();

for (int i = 0; i < parameters.Length; i++)

{

if (parameters[i].ParameterType.Name == param)

{

Console.WriteLine(method.Name);

fullInfo += method.Name + "\n"; }

}

}

}

public static void Invoke(string ClassName, string MethodName)

{

Type myType = Type.GetType("Lab11." + ClassName);

foreach (MethodInfo method in myType.GetMethods())

{

if (method.Name == MethodName)

{

object obj = Activator.CreateInstance(myType, "Попугай");

method.Invoke(obj, new object[] {ReadFromFile()});

}

}

}

public static T Create<T> ()

{

Type t = typeof(T);

object obj = Activator.CreateInstance(t, "Попугай");

return (T)obj;

}

}

internal class Program

{

static void Main(string[] args)

{

Reflector.BuildInf();

Reflector.ConstrInf("Tiger");

Reflector.MethodInf("Tiger");

Reflector.FieldInf("Tiger");

Reflector.ProptInf("Tiger");

Reflector.InterInf("Tiger");

Console.ReadKey();

Reflector.MethodForParamInf("Parrot");

Reflector.WriteToFile();

Reflector.Invoke("Parrot", "Speak");

object par = Reflector.Create<Parrot>();

Type f = par.GetType();

Console.WriteLine(f.Name);

}

}

}

Файловая система

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.IO.Compression;

using System.IO;

namespace Lab12

{

public static class LogInfo

{

public static void WriteLogInfo()

{

string logPath = Path.GetFullPath("D:/ididirinfo.txt");

try

{

using (StreamWriter sw = new StreamWriter(logPath, false, Encoding.Default))

{

sw.WriteLine("<=========================================== iDILog ===================================================>");

sw.WriteLine($"Имя файла лога: {Path.GetFileName(logPath)}");

sw.WriteLine($"Полный путь лога: {logPath}");

sw.WriteLine($"Время записи лога: {DateTime.Now}");

}

}

catch (Exception e)

{

Console.WriteLine(e.Message);

}

}

public static void WriteInLog(string message)

{

string logPath = Path.GetFullPath("D/ididirinfo.txt");

try

{

using (StreamWriter sw = new StreamWriter(logPath, true, Encoding.Default))

{

sw.WriteLine(message);

}

}

catch (Exception e)

{

Console.WriteLine(e.Message);

}

}

public static string ReadLog()

{

string logPath = Path.GetFullPath("D/ididirinfo.txt");

try

{

StreamReader sr = new StreamReader(logPath);

return sr.ReadToEnd();

}

catch (Exception e)

{

Console.WriteLine(e.Message);

return String.Empty;

}

}

}

static class IDILog

{

public static void WriteInfo(string info)

{

StreamWriter sw = File.AppendText("D:/лабы ООП/Lab12/idilogfile.txt");

sw.WriteLine(info);

sw.WriteLine();

sw.Close();

}

}

class IDIDiskInfo

{

DriveInfo[] drivers = DriveInfo.GetDrives();

public void DriversInfo()

{

try

{

foreach (DriveInfo drive in drivers)

{

Console.WriteLine(drive.Name);

Console.WriteLine(drive.TotalSize / 1024 / 1024 / 1024 + "Гб");

Console.WriteLine(drive.AvailableFreeSpace / 1024 / 1024 / 1024 + "Гб");

Console.WriteLine(drive.DriveFormat);

}

}

catch

{

}

}

}

class IDIFileInfo

{

string dirname = "D:/лабы ООП/Lab12/Test.txt";

public void FileInformation()

{

FileInfo file = new FileInfo(dirname);

Console.WriteLine(file.FullName);

Console.WriteLine(file.Name);

Console.WriteLine(file.Length);

Console.WriteLine(file.CreationTime);

IDILog.WriteInfo("Просотрен файл: " + file.Name + ", Путь: " + file.FullName + ", Размер: " + file.Length + ", Время создания: " + file.CreationTime);

}

}

class IDIDirInfo

{

string dirname = "D:/лабы ООП/Lab12/TestFolder/MyFiles";

public void DirInformation()

{

Console.WriteLine("Директории:");

string[] dirs = Directory.GetDirectories(dirname);

foreach (string dir in dirs)

{

DirectoryInfo CurrentDir = new DirectoryInfo(dir);

Console.WriteLine(CurrentDir.Name);

}

Console.WriteLine();

Console.WriteLine("Файлы: ");

string[] files = Directory.GetFiles(dirname);

foreach (string file in files)

{

FileInfo CurrentFile = new FileInfo(file);

Console.WriteLine(file);

Console.WriteLine(CurrentFile.Name);

Console.WriteLine(CurrentFile.Length);

Console.WriteLine(CurrentFile.CreationTime);

Console.WriteLine();

}

}

}

class IDIFileManager

{

string DirPath = "D:/лабы ООП/Lab12/TestFolder";

string DiskPath = "D:/";

public void SaveInfoInTxt()

{

string[] dirs = Directory.GetDirectories(DiskPath);

DirectoryInfo dirInfo = new DirectoryInfo(DirPath);

dirInfo.CreateSubdirectory("IDIInspect");

dirInfo = new DirectoryInfo(DirPath + "/IDIInspect");

string logPath = Path.GetFullPath("D:/лабы ООП/Lab12/TestFolder/IDIInspect");

try

{

using (StreamWriter sw = new StreamWriter(DirPath + "/ididirinfo.txt"))

{

sw.WriteLine("<=========================================== iDILog ===================================================>");

sw.WriteLine($"Имя директории: {dirInfo.Name}");

sw.WriteLine($"Полный путь директории: {logPath}");

sw.WriteLine($"Время создания директории: {dirInfo.CreationTime}");

}

}

catch (Exception e)

{

Console.WriteLine(e.Message);

}

try

{

File.Copy(DirPath + "/ididirinfo.txt", DirPath + "/ididirinfo2.txt");

}

catch

{

}

File.Delete(DirPath + "/ididirinfo.txt");

}

public void MoveDir()

{

DirectoryInfo dirInfo = new DirectoryInfo(DirPath);

dirInfo.CreateSubdirectory("IDIFiles");

string[] txtList = Directory.GetFiles("D:/лабы ООП/Lab12/TestFolder/MyFiles", "\*.txt");

FileInfo FileInf;

foreach (string file in txtList)

{

FileInf = new FileInfo(file);

File.Copy(file, Path.Combine(DirPath + "/IDIFiles", FileInf.Name), true);

IDILog.WriteInfo("Создан файл: " + FileInf.Name + ", Путь: " + file + ", Размер: " + FileInf.Length + ", Время создания: " + FileInf.CreationTime);

}

Directory.Move(DirPath + "/IDIFiles", DirPath + "/IDIInspect/IDIFiles");

}

public void Compress()

{

string sourseFile = "D:/лабы ООП/Lab12/TestFolder/IDIInspect/IDIFiles";

string compressedFile = "D:/лабы ООП/Lab12/TestFolder/IDIInspect/AR.rar";

ZipFile.CreateFromDirectory(sourseFile, compressedFile);

ZipFile.ExtractToDirectory(compressedFile, "D:/лабы ООП/Lab12/TestFolder/IDIInspect/UnAr");

}

}

class ReadIdiLog

{

public void FindWrite (string day)

{

string[] FileText = File.ReadAllLines("D:/лабы ООП/Lab12/idilogfile.txt");

foreach (string str in FileText)

{

if(str.Contains("Время создания: " + day)) Console.WriteLine(str);

}

}

public void CountWrite()

{

int count = 0;

string[] FileText = File.ReadAllLines("D:/лабы ООП/Lab12/idilogfile.txt");

foreach (string str in FileText)

{

if (str != "") count++;

}

Console.WriteLine("Количество записей: " + count);

}

public void DelPart (string time)

{

string[] FileText = File.ReadAllLines("D:/лабы ООП/Lab12/idilogfile.txt");

string newInfo = "";

foreach (string str in FileText)

{

if (str.Contains(time + ":")) newInfo += str + "\n\n";

}

StreamWriter sw = new StreamWriter("D:/лабы ООП/Lab12/idilogfile.txt");

sw.WriteLine(newInfo);

sw.Close();

}

}

internal class Program

{

static void Main(string[] args)

{

IDIDiskInfo drivers = new IDIDiskInfo();

drivers.DriversInfo();

Console.ReadKey();

IDIFileInfo file = new IDIFileInfo();

file.FileInformation();

Console.ReadKey();

IDIDirInfo dirs = new IDIDirInfo();

dirs.DirInformation();

Console.ReadKey();

IDIFileManager manager = new IDIFileManager();

manager.SaveInfoInTxt();

manager.MoveDir();

manager.Compress();

ReadIdiLog lg = new ReadIdiLog();

lg.FindWrite("29");

lg.CountWrite();

lg.DelPart("14");

}

}

}

Сериализация

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Runtime.Serialization.Formatters.Binary;

using System.IO;

using System.Xml.Serialization;

using System.Xml;

using System.Xml.Linq;

using System.Runtime.ConstrainedExecution;

using Newtonsoft.Json;

using System.Runtime.Serialization.Formatters.Soap;

using System.Runtime.Serialization;

using System.Runtime.InteropServices.ComTypes;

namespace Lab13

{

static class CustomSerializer

{

public static void XmlSer(Lion[] an)

{

XmlSerializer xml = new XmlSerializer(typeof(Lion[]));

using (FileStream fs = new FileStream("Animal.xml", FileMode.OpenOrCreate))

{

xml.Serialize(fs, an);

Console.WriteLine("Объект успешно сериализован в XML файл!");

}

using (FileStream fs = new FileStream("Animal.xml", FileMode.OpenOrCreate))

{

Lion[] lions = (Lion[])xml.Deserialize(fs);

Console.WriteLine("Объект десериализован с XML файла!");

foreach (Lion lion in lions)

{

Console.WriteLine($"Имя: {lion.Name} --- здоровье: {lion.HP}");

}

}

Console.WriteLine();

}

public static void BinSer(Lion[] an)

{

BinaryFormatter Bin = new BinaryFormatter();

using (FileStream fs = new FileStream("Animal.dat", FileMode.OpenOrCreate))

{

Bin.Serialize(fs, an);

Console.WriteLine("Объект успешно сериализован в BIN!");

}

using (FileStream fs = new FileStream("Animal.dat", FileMode.OpenOrCreate))

{

Lion[] lions = (Lion[])Bin.Deserialize(fs);

Console.WriteLine("Объект десериализован с BIN!");

foreach (Lion lion in lions)

{

Console.WriteLine($"Имя: {lion.Name} --- здоровье: {lion.HP}");

}

}

Console.WriteLine();

}

public static void SoapSer(Lion[] an)

{

SoapFormatter Soap = new SoapFormatter();

using (FileStream fs = new FileStream("Animal.soap", FileMode.OpenOrCreate))

{

Soap.Serialize(fs, an);

Console.WriteLine("Объект успешно сериализован в SOAP файл!");

}

using (FileStream fs = new FileStream("Animal.soap", FileMode.OpenOrCreate))

{

Lion[] lions = (Lion[])Soap.Deserialize(fs);

Console.WriteLine("Объект десериализован с SOAP файла!");

foreach (Lion lion in lions)

{

Console.WriteLine($"Имя: {lion.Name} --- здоровье: {lion.HP}");

}

}

Console.WriteLine();

}

public static void JSONser(Lion[] an)

{

string json = JsonConvert.SerializeObject(an);

Console.WriteLine("Объект успешно сериализован в JSON!");

Lion[] restoredPerson = JsonConvert.DeserializeObject<Lion[]>(json);

Console.WriteLine("Объект десериализован с JSON!");

foreach (Lion lion in restoredPerson)

{

Console.WriteLine($"Имя: {lion.Name} --- здоровье: {lion.HP}");

}

Console.WriteLine();

}

}

static class XmlSelectors

{

public static void Do()

{

XmlDocument xDoc = new XmlDocument();

xDoc.Load("Animal.xml");

XmlElement xRoot = xDoc.DocumentElement;

XmlNodeList childnodes = xRoot.SelectNodes("Lion");

foreach (XmlNode n in childnodes)

Console.WriteLine(n.OuterXml);

Console.ReadLine();

}

}

static class LINQtoXML

{

public static void Do(Lion[] lions)

{

#region linq to xml

XDocument anything = new XDocument();

XElement Root = new XElement("Lions");

foreach (Lion lion in lions)

{

XElement El = new XElement("Lion");

XAttribute xAttribute = new XAttribute("Name", lion.Name);

XElement Health = new XElement("Name", lion.HP);

El.Add(xAttribute);

El.Add(Health);

Root.Add(El);

}

anything.Add(Root);

anything.Save("Anything.xml");

#endregion

}

}

internal class Program

{

static void Main(string[] args)

{

Lion[] lions = { new Lion("Лёва"), new Lion("Лев Петя"), new Lion("Котик") };

CustomSerializer.XmlSer(lions);

CustomSerializer.BinSer(lions);

CustomSerializer.SoapSer(lions);

CustomSerializer.JSONser(lions);

XmlSelectors.Do();

LINQtoXML.Do(lions);

}

}

}

Потоки, процессы, домены

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Diagnostics;

using System.Reflection;

using System.Threading;

using System.IO;

namespace Lab14

{

static class Processes

{

public static void ShowProcesses()

{

foreach (Process process in Process.GetProcesses())

{

try

{

Console.WriteLine($"ID: {process.Id} Name: {process.ProcessName} Prioritet: {process.BasePriority} Time: {process.StartTime}");

}

catch

{

continue;

}

}

}

}

static class DomainInfo

{

public static void Info()

{

AppDomain domain = AppDomain.CurrentDomain;

Console.WriteLine($"Name: {domain.FriendlyName}");

Console.WriteLine($"Путь: {domain.BaseDirectory}");

Assembly[] assemblies = domain.GetAssemblies();

foreach (Assembly asm in assemblies)

Console.WriteLine(asm.GetName().Name);

AppDomain NewDom = AppDomain.CreateDomain("Gym");

NewDom.ExecuteAssembly("Build.exe");

Console.WriteLine($"Name: {NewDom.FriendlyName}");

Console.WriteLine($"Путь: {NewDom.BaseDirectory}");

AppDomain.Unload(NewDom);

}

}

static class MyThread

{

public static void MyFirstThread()

{

Console.Write("Введите n: ");

int n = int.Parse(Console.ReadLine());

Thread thread = new Thread(new ParameterizedThreadStart(SimpleNumbers));

thread.Name = "Мой любимый поток";

object o = n;

thread.Start(o);

}

public static void SimpleNumbers(object o)

{

int n = (int)o;

StreamWriter sr = new StreamWriter("D:/лабы ООП/Lab14/Numbs.txt");

Console.Write("Name: ");

Console.WriteLine(Thread.CurrentThread.Name);

Console.WriteLine(Thread.CurrentThread.Priority);

for (int i = 1; i <= n; i++)

{

Console.WriteLine(i);

sr.WriteLine(i);

Thread.Sleep(100);

}

Console.WriteLine("Finish");

sr.Close();

}

}

static class TwoThreads

{

private static object locker = new object();

public static void MyFirstThread()

{

Thread thread1 = new Thread(new ParameterizedThreadStart(Numbers));

Thread thread2 = new Thread(new ParameterizedThreadStart(Numbers));

thread1.Name = "Чётные";

thread2.Name = "Нечётные";

thread1.Priority = ThreadPriority.BelowNormal;

thread2.Priority = ThreadPriority.Normal;

Console.WriteLine("Введите n: ");

int n = int.Parse(Console.ReadLine());

object o = n;

thread1.Start(o);

thread2.Start(o);

}

public static void Numbers(object n)

{

int number = (int)n;

lock (locker)

{

StreamWriter sr = File.AppendText("D:/лабы ООП/Lab14/Numbs2.txt");

for (int i = 1; i <= number; i++)

{

if (i % 2 == 0 && Thread.CurrentThread.Name == "Чётные")

{

Console.WriteLine(i);

sr.WriteLine(i);

Thread.Sleep(200);

}

if (i % 2 != 0 && Thread.CurrentThread.Name == "Нечётные")

{

Console.WriteLine(i);

sr.WriteLine(i);

Thread.Sleep(400);

}

}

sr.Close();

}

}

}

static class Timerr

{

public static void MyTimer()

{

int num = 4;

TimerCallback tm = new TimerCallback(Count);

Timer timer = new Timer(tm, num, 5500, 2000);

Console.ReadLine();

}

public static void Count(object obj)

{

int x = (int)obj;

for (int i = 1; i < x; i++)

{

Console.WriteLine($"Отработал Timer: {i}");

}

}

}

class Program

{

static void Main(string[] args)

{

Processes.ShowProcesses();

Console.WriteLine();

DomainInfo.Info();

Console.WriteLine();

MyThread.MyFirstThread();

Console.WriteLine();

TwoThreads.MyFirstThread();

Console.WriteLine();

Timerr.MyTimer();

Console.WriteLine();

}

}

}

Task

using System;

using System.Collections.Concurrent;

using System.Diagnostics;

using System.Threading;

using System.Threading.Tasks;

namespace Lab15

{

internal class Program

{

private static object locker = new object();

private static int[] Eratosthen()

{

int n = 300;

int[] arr = new int[n+1];

for (var i = 2; i < n; i++) arr[i] = 1;

for (var i = 2; i \* i <= n; i++)

{

if (arr[i] == 1)

{

for (var j = i \* i; j <= n; j += i)

{

arr[j] = 0;

}

}

}

return arr;

}

private static int[] EratosthenCancellationToken(object cancel)

{

int n = 300;

var token = (CancellationToken)cancel;

int[] arr = new int[n + 1];

for (var i = 2; i < n; i++) arr[i] = 1;

for (var i = 2; i \* i <= n; i++)

{

if (token.IsCancellationRequested)

{

Console.WriteLine($"Таск №{Task.CurrentId} прерван токеном");

return null;

}

if (arr[i] == 1)

{

for (var j = i \* i; j <= n; j += i)

{

arr[j] = 0;

}

}

}

return arr;

}

private static void StoragePrint(BlockingCollection<string> stor)

{

lock(locker)

{

Console.WriteLine("\nСклад: ");

foreach (var item in stor)

{

Console.WriteLine(item);

}

Console.WriteLine("\n");

}

}

private static async void PrintResult()

{

long sum = 0;

await Task.Run(() =>

{

for (var i = 0; i < 1000000000; i++)

{

if (i % 2 == 0) sum += i;

}

});

Console.WriteLine("Результат: " + sum);

}

static void Main(string[] args)

{

//1

var watcher = new Stopwatch();

var task = new Task<int[]>(Eratosthen);

Console.WriteLine("Task id: " + task.Id);

Console.WriteLine("Task status: " + task.Status);

task.Start();

watcher.Start();

Console.WriteLine("Task id: " + task.Id);

Console.WriteLine("Task status: " + task.Status);

task.Wait();

watcher.Stop();

Console.WriteLine($"\nВремя выполнения задачи: {watcher.ElapsedMilliseconds} мс");

//2

var tokenSource = new CancellationTokenSource();

var task2 = new Task<int[]>(EratosthenCancellationToken, tokenSource.Token);

task2.Start();

tokenSource.Cancel();

//3, 4

int a = 3;

int b = 14;

int c = 6;

var first = Task.Run(() => a + b);

var second = Task.Run(() => a \* c);

var third = Task.Run(() => b - c);

Task.WhenAll(first, second, third).ContinueWith(res => Console.WriteLine("Result:" + first.Result + second.Result + third.Result));

var awaiter = first.GetAwaiter();

awaiter.OnCompleted(() =>

{

Task.Run(() => Console.WriteLine("Квадрат из первого числа: " + Math.Pow(first.Result, 2)));

});

//5

var arr1 = new int[1000000];

var arr2 = new int[1000000];

var arr3 = new int[1000000];

var sw = new Stopwatch();

var forResult = Parallel.For(1, 1000000, i =>

{

sw.Start();

arr1[i] = i;

arr2[i] = i;

arr3[i] = i;

});

sw.Stop();

Console.WriteLine($"Parallel for заполнил массивы за: {sw.ElapsedMilliseconds} мс");

sw.Restart();

for (int i = 0; i < 1000000; i++)

{

arr1[i] = i;

arr2[i] = i;

arr3[i] = i;

}

sw.Stop();

Console.WriteLine($"For заполнил массивы за {sw.ElapsedMilliseconds} мс");

sw.Restart();

var foreachResult = Parallel.ForEach(arr1, el => arr2[el] = 0);

sw.Stop();

Console.WriteLine($"Parallel foreach заполнил массив нулями за {sw.ElapsedMilliseconds} мс");

sw.Restart();

foreach (var i in arr3)

{

arr3[i] = 0;

}

sw.Stop();

Console.WriteLine($"Foreach заполнил массив нулями за {sw.ElapsedMilliseconds} мс");

//6

Parallel.Invoke(

() =>

{

for (var i = 0; i < 1000000; i++)

{

arr1[i] = i;

}

},

() => {

for (var i = 0; i < 1000000; i++)

{

arr2[i] = i;

}

},

() => {

for (var i = 0; i < 1000000; i++)

{

arr3[i] = i;

}

});

//7, 8

var storage = new BlockingCollection<string>(5);

Task provider1 = Task.Run(() =>

{

for (int i = 0; i < 5; i++)

{

Thread.Sleep(1500);

if(storage.TryAdd("Холодильник")) StoragePrint(storage);

}

});

Task provider2 = Task.Run(() =>

{

for (int i = 0; i < 5; i++)

{

Thread.Sleep(1500);

if(storage.TryAdd("Стиральная машина")) StoragePrint(storage);

}

});

Task provider3 = Task.Run(() =>

{

for (int i = 0; i < 5; i++)

{

Thread.Sleep(1500);

if(storage.TryAdd("Видеокарта")) StoragePrint(storage);

}

});

Task provider4 = Task.Run(() =>

{

for (int i = 0; i < 5; i++)

{

Thread.Sleep(1500);

if(storage.TryAdd("Ноутбук")) StoragePrint(storage);

}

});

Task provider5 = Task.Run(() =>

{

for (int i = 0; i < 5; i++)

{

Thread.Sleep(1500);

if(storage.TryAdd("Наушники")) StoragePrint(storage);

}

});

Task[] Buyers = new Task[10]

{

Task.Run(() =>

{

for(int i = 0; i < 5; i++)

{

Thread.Sleep(900);

storage.Take();

}

}),

Task.Run(() =>

{

for(int i = 0; i < 5; i++)

{

Thread.Sleep(900);

storage.Take();

}

}),

Task.Run(() =>

{

for(int i = 0; i < 5; i++)

{

Thread.Sleep(900);

storage.Take();

}

}),

Task.Run(() =>

{

for(int i = 0; i < 5; i++)

{

Thread.Sleep(900);

storage.Take();

}

}),

Task.Run(() =>

{

for(int i = 0; i < 5; i++)

{

Thread.Sleep(900);

storage.Take();

}

}),

Task.Run(() =>

{

for(int i = 0; i < 5; i++)

{

Thread.Sleep(900);

storage.Take();

}

}),

Task.Run(() =>

{

for(int i = 0; i < 5; i++)

{

Thread.Sleep(900);

storage.Take();

}

}),

Task.Run(() =>

{

for(int i = 0; i < 5; i++)

{

Thread.Sleep(900);

storage.Take();

}

}),

Task.Run(() =>

{

for(int i = 0; i < 5; i++)

{

Thread.Sleep(900);

storage.Take();

}

}),

Task.Run(() =>

{

for(int i = 0; i < 5; i++)

{

Thread.Sleep(900);

storage.Take();

}

})

};

PrintResult();

Console.WriteLine("Ожидаем расчета: ");

Console.ReadKey();

}

}

}

Pattern

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Runtime.Serialization.Formatters.Binary;

using System.Runtime.Serialization;

using System.Text;

using System.Threading.Tasks;

namespace Lab17\_18

{

public class Application

{

// public int Size { get; set; }

public string WorkType { get; set; }

public int Time { get; set; }

public int Size { get; set; }

public Application(string type, int time, int size)

{

Size = size;

WorkType = type;

Time = time;

}

public Application()

{

Size = 0;

WorkType = "Нет работы";

Time = 0;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab17\_18

{

public interface IFactory

{

Application CreateApplication(Tenant tenant);

}

public class Builder : IFactory

{

public Application CreateApplication(Tenant tenant)

{

return new Application(tenant.Specialization, tenant.Time, tenant.Size);

}

public Application CreateEmptyApplication()

{

return new Application();

}

}

}

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Reflection;

using System.Runtime.Serialization.Formatters.Binary;

using System.Runtime.Serialization;

using System.Text;

using System.Threading.Tasks;

using System.Xml.Linq;

using System.Security.Cryptography;

using System.Runtime.InteropServices.ComTypes;

namespace Lab17\_18

{

public interface IBrigade

{

IBrigade Clone();

object DeepCopy();

void PrintNames();

}

public interface IWorkingBrigade

{

void Update();

}

public class Dispetcher

{

UnemployedHistory history = new UnemployedHistory();

[Serializable]

private class Brigade : IBrigade, IWorkingBrigade

{

public string WorkType { get; set; }

public int Time { get; set; }

public int Size { get; set; }

public List<string> Names = new List<string>();

IWork work;

public Brigade(string type, int time, int size, string name)

{

Size = size;

WorkType = type;

Time = time;

Names.Add(name);

}

public void AddWork (IWork w)

{

work = w;

work.Reg(this);

}

public void StopWork()

{

work.Remove(this);

work = null;

}

public void AddTeanet(string name)

{

Names.Add(name);

}

public void PrintNames()

{

foreach (string name in Names)

{

Console.Write(name + ", ");

}

}

public void Update()

{

Console.Write("Бригада из "); PrintNames(); Console.WriteLine(" отработала!");

}

public IBrigade Clone()

{

return this.MemberwiseClone() as IBrigade;

}

public object DeepCopy()

{

object figure = null;

using (MemoryStream tempStream = new MemoryStream())

{

BinaryFormatter binFormatter = new BinaryFormatter(null,

new StreamingContext(StreamingContextStates.Clone));

binFormatter.Serialize(tempStream, this);

tempStream.Seek(0, SeekOrigin.Begin);

figure = binFormatter.Deserialize(tempStream);

}

return figure;

}

}

interface IUnemployed

{

void OutNames();

}

private class Unemployed : Brigade, IUnemployed

{

public Unemployed(string type, int time, int size, string name) : base(type, time, size, name)

{

}

public void OutNames()

{

foreach (string name in Names)

{

Console.Write(name + ", ");

}

}

public UnemployedMemento SaveState()

{

UnemployedMemento memento = new UnemployedMemento();

foreach (string name in Names)

{

memento.Names.Add(name);

}

return memento;

}

public void RestoreState(UnemployedHistory history)

{

Console.WriteLine("История зполнения тунеядцев!");

for (int i = 0; i < history.History.Count; i++)

{

Console.Write($"{i} Запись: ");

foreach (string name in history.History.ElementAt(i).Names)

{

Console.Write(name + ", ");

}

Console.WriteLine();

}

}

}

private class UnemAdapter : IBrigade

{

protected Unemployed unemployed;

public UnemAdapter(Unemployed un)

{

this.unemployed = un;

}

public void PrintNames()

{

unemployed.OutNames();

}

public IBrigade Clone()

{

return this.MemberwiseClone() as IBrigade;

}

public object DeepCopy()

{

object figure = null;

using (MemoryStream tempStream = new MemoryStream())

{

BinaryFormatter binFormatter = new BinaryFormatter(null,

new StreamingContext(StreamingContextStates.Clone));

binFormatter.Serialize(tempStream, this);

tempStream.Seek(0, SeekOrigin.Begin);

figure = binFormatter.Deserialize(tempStream);

}

return figure;

}

}

private class BrigadeNames

{

public void ShowNames(IBrigade brigade)

{

brigade.PrintNames();

}

}

private Unemployed unemployed;

private BrigadeNames Names = new BrigadeNames();

private UnemAdapter unAdapter;

private Work work = new Work();

private List<Brigade> BrigadeList = new List<Brigade>();

public void PrintBrigades()

{

int i = 1;

Console.WriteLine("\n\nCФОРМОРОВАНЫЕ БРИГАДЫ");

foreach (Brigade brigade in BrigadeList)

{

Console.WriteLine($"---------------------- Бригада {i} ---------------------");

Console.WriteLine($"Вид работы: {brigade.WorkType}");

Console.WriteLine($"Масштаб работы: {brigade.Size}");

Console.WriteLine($"Время работы: {brigade.Time}");

//Console.Write($"Имена квартиросъёмщиков: "); brigade.PrintNames();

Names.ShowNames(brigade);

Console.WriteLine("\n------------------------------------------------------");

i++;

}

if (unemployed != null)

{

Console.WriteLine($"---------------------- Тунеядцы ---------------------");

unAdapter = new UnemAdapter(unemployed);

Names.ShowNames(unAdapter);

Console.WriteLine();

}

}

public void HistoryUnemployed()

{

//unemployed.RestoreState(history);

Console.WriteLine("История зполнения тунеядцев!");

for (int i = 0; i < history.History.Count; i++)

{

Console.Write($"{i} Запись: ");

foreach (string name in history.History.ElementAt(i).Names)

{

Console.Write(name + ", ");

}

Console.WriteLine();

}

}

public void PushHistory()

{

history.History.Add(unemployed.SaveState());

}

public void SentApplication(Builder build, Tenant tenat)

{

Application Form = build.CreateApplication(tenat);

foreach (Brigade brigade in BrigadeList)

{

if (Form.WorkType == brigade.WorkType && Form.Time == brigade.Time && Form.Size == brigade.Size)

{

brigade.AddTeanet(tenat.Name);

return;

}

}

BrigadeList.Add(new Brigade(Form.WorkType, Form.Time, Form.Size, tenat.Name));

}

public void SentEmptyApplication(Builder build)

{

Application Form = build.CreateEmptyApplication();

Console.Write("Введите ваше имя: ");

string Name = Console.ReadLine();

Console.WriteLine("----------- Ваши данные сохранены! -----------\n");

if (unemployed == null) unemployed = new Unemployed(Form.WorkType, Form.Time, Form.Size, Name);

else unemployed.Names.Add(Name);

}

public void CopyBrigade(int i)

{

Brigade CloneBrigade = BrigadeList.ElementAt(i);

BrigadeList.Add(CloneBrigade.DeepCopy() as Brigade);

}

public void AddBrigadeWork(int i)

{

BrigadeList.ElementAt(i).AddWork(work);

}

public void RemoveBrigadeWork(int i)

{

BrigadeList.ElementAt(i).StopWork();

}

public void GoToWork()

{

work.Working();

}

//Singelton

private static Dispetcher instance;

private Dispetcher() { }

public static Dispetcher getInstance()

{

if (instance == null)

instance = new Dispetcher();

return instance;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab17\_18

{

public class UnemployedMemento

{

public List<string> Names = new List<string>();

}

public class UnemployedHistory

{

public List<UnemployedMemento> History;

public UnemployedHistory()

{

History = new List<UnemployedMemento>();

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab17\_18

{

public abstract class Tenant

{

public TenantState lifestyle { get; set; }

public void Training()

{

if (lifestyle == TenantState.GO\_TO\_GYM) Size += 2;

if (lifestyle == TenantState.LAYING\_ON\_SOFA) Size++;

}

public void Relax()

{

if (lifestyle == TenantState.GO\_TO\_GYM) Size--;

if (lifestyle == TenantState.LAYING\_ON\_SOFA) Size -= 2;

}

public string Name { get; set; }

public string Specialization { get; set; }

public int Size { get; set; }

public int Time { get; set; }

public Tenant (string WorkType, string name, TenantState state)

{

Specialization = WorkType;

Name = name;

lifestyle = state;

if(lifestyle == TenantState.LAYING\_ON\_SOFA) Size = 1;

if(lifestyle == TenantState.GO\_TO\_GYM) Size = 3;

}

public Tenant(int time, string name, string WorkType, int size, TenantState state)

{

Time = time;

Name = name;

Specialization = WorkType;

Size = size;

}

}

public class Cleaner : Tenant

{

public Cleaner(string name, TenantState state) : base("Уборка", name, state)

{

}

}

public class Painter : Tenant

{

public Painter(string name, TenantState state) : base("Красить стены", name, state)

{

}

}

public abstract class WorkingShift : Tenant

{

protected Tenant tenant;

public WorkingShift(int time, Tenant tenant) : base(time, tenant.Name, tenant.Specialization, tenant.Size, tenant.lifestyle)

{

this.tenant = tenant;

}

}

public class AM : WorkingShift

{

public AM(Tenant tenant) : base(18, tenant)

{ }

}

public class PM : WorkingShift

{

public PM(Tenant tenant) : base(9, tenant)

{ }

}

public enum TenantState

{

LAYING\_ON\_SOFA,

GO\_TO\_GYM

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab17\_18

{

public interface IWork

{

void Reg(IWorkingBrigade b);

void Remove(IWorkingBrigade b);

void NotifyObservers();

}

public class Work : IWork

{

public int WorkProc;

List<IWorkingBrigade> brigades;

public Work ()

{

brigades = new List<IWorkingBrigade>();

WorkProc = 0;

}

public void Reg(IWorkingBrigade b)

{

brigades.Add(b);

}

public void Remove(IWorkingBrigade b)

{

brigades.Remove(b);

}

public void NotifyObservers()

{

foreach (IWorkingBrigade b in brigades)

{

b.Update();

}

}

public void Working()

{

WorkProc += 10 \* brigades.Count;

NotifyObservers();

Console.WriteLine($"Работа выполнена на {WorkProc}%!");

}

}

}

class Client

{

void Main()

{

Subject subject = new Proxy();

subject.Request();

}

}

abstract class Subject

{

public abstract void Request();

}

class RealSubject : Subject

{

public override void Request()

{}

}

class Proxy : Subject

{

RealSubject realSubject;

public override void Request()

{

if (realSubject == null)

realSubject = new RealSubject();

realSubject.Request();

}

}